BWDSP10x 上地址和数据谓词执行的编译优化①

樊永朝[1,2], 郑启龙[1,2], 耿　锐[3], 王向前[3], 王　昊[3]

[1](中国科学技术大学 安徽省高性能计算重点实验室, 合肥 230027)
[2](中国科学技术大学 计算机科学与技术学院, 合肥 230027)
[3](中国电子科技集团公司 第三十八研究所, 合肥 230088)

摘　要: 传统的谓词优化技术是在冯•诺伊曼体系结构计算机上实施的, 仅对数据流进行优化, 并没有考虑哈佛体系结构下指令和数据分开的情况. BWDSP10x 是指令和数据分开的哈佛体系结构, 它支持超长指令字, 不仅提供了对数据谓词执行的支持也提供了对地址谓词执行的支持. 特此提出了一种在区域上对两种谓词模式优化支持的方法, 在进行两种比较之前, 通过判断比较操作的两个操作数类型来分别实施两种模式的谓词优化, 使得对地址的比较不用传输到通用寄存器中. 实验结果表明该优化方法能显著地节省 CPU 的时间和带宽, 大大减少了分支指令, 使程序性能提高了 28.4%.

关键词: 地址谓词执行; 数据谓词执行; 区域; 编译优化

Compilation Optimization of Address and Data Predicated Execution on BWDSP10x

FAN Yong-Chao[1,2], ZHENG Qi-Long[1,2], GENG Rui[3], WANG Xiang-Qian[3], WANG Hao[3]

[1](Anhui High Performance Computing Key Laboratory, University of Science and Technology of China, Hefei 230027, China)
[2](School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)
[3](No. 38th Research Institute, China Electronics Technology Group Corporation, Hefei 230088, China)

Abstract: The traditional predicate optimization technique is based on the Von Neumann architecture, which considers the data flow optimization only. However, BWDSP10x is based on the Harvard architecture, which data and instructions are physically separated. It provides VLIW and supports not only data predicated execution but also address predicated execution. Hence, we present an optimization method for the two predicated execution technology based on the region. In this method, types of the two operands of comparison operation will be identified before the two kinds of operations are executed, and when addresses are compared, the two operands don't need to transfer to general registers. Experimental result shows that the optimization method can highly reduce the time and bandwidth of CPU, and reduce large numbers of branch instructions. The performance of programs tested is increased by 28.4 percent after the optimization.

Key words: address predicated execution; data predicated execution; region; compilation optimization

指令在执行中要经历取指、译码、执行、存取等阶段. 现在的处理器中大多都有多个执行单元, 从而能使上述过程实现指令流水, 达到指令级并行的效果. 但由于程序中条件分支的存在, 使指令流水停顿, 减弱了指令流水的效果. 为了减弱条件分支对指令流水的影响, 传统的优化方法有软件流水[1]、指令调度[2]、分支预测[3]等, 软件流水和指令调度通过循环展开, 调整指令之间的次序来减弱分支指令对指令流水的影响, 但没有从根本上消除条件分支. 分支预测使用分支预测器来猜测哪条分支将会被执行, 它的缺点是当发生分支误预测时, 流水线的性能大大降低.

解决条件分支的另一种方法是谓词执行[4], 利用条件转换[5]使所有可能的分支路径都被编码, 在实际执行中使一些指令执行, 另一些指令不执行. 谓词执行的基本思想是每条指令和一个谓词相关, 只有当谓词为真时指令才被执行, 这样便消除了条件分支, 使

① 基金项目:"核高基"重大专项(2012ZX01034-00-001)
收稿时间:2016-03-22;收到修改稿时间:2016-06-12　[doi:10.15888/j.cnki.csa.005573]

被条件分支分割的多个基本块合为一个基本块, 虽然增加了基本块的长度, 但和减少的分支停顿相比这点代价很小. 传统的谓词优化[6]仅仅针对冯•诺伊曼体系结构, 对地址和数据的谓词优化都放在通用寄存器中比较, 而 BWDSP10x 的哈佛体系结构指令和数据是分开的, 地址和数据的比较操作都采用一种模式不能充分利用处理器的资源.

本文基于 BWDSP10x 体系结构提出了一种对地址和数据比较都进行谓词优化的方法, 对条件分支中地址和数据的比较分开进行, 减少了不必要的数据传输, 提高了带宽利用率, 使流水线停顿的次数大大减少, 从而提高了程序性能.

1 编译器框架

1.1 BWDSP10x 体系结构

BWDSP10x 处理器[7]是一款 32 位静态超标量处理器, 采用 16 发射、SIMD(单指令流, 多数据流)架构. BWDSP10x 处理器内核有 4 个运算宏, 每个运算宏由 8 个算术逻辑单元、4 个乘法器、2 个移位器、1 个超算器、1 个 8 位数据谓词寄存器以及 1 个通用寄存组组成. 运算宏之间通过宏间传输总线通信. 内部运算单元和存储器之间的数据交换所需的存储器地址主要是通过内部若干个地址发生器来提供. BWDSP10x 内部有三个地址发生器(U/V/W), 地址发生器由三部分构成: 瞬时地址运算器、地址更新运算器和 16 个寄存器组构成的地址寄存器组, 每个地址寄存器位宽为 32 位.

1.2 BWDSP10x 编译器谓词优化框架

BWDSP10x 的编译器基于开源编译器 OPEN64, OPEN64 前端是 GCC, 后端构建在 WHIRL 中间语言的基础上, OPEN64 是为 Intel IA64 提供编译优化支持的[5]. BWDSP10x 的谓词模块在后端代码生成(CG)阶段, 在 CG 阶段开始后, 以中间语言 VL WHIRL 作为输入, 经过 CG expansion、生成 region tree 信息后开始以区域[8]为单元执行谓词优化. 图 1 描述了 BWDSP10x 谓词执行优化框架, 其中谓词定义指令生成、谓词执行指令生成构成了本文谓词优化算法.

2 添加机器描述

现阶段为了支持多种目标体系结构, 许多编译器基础设施都采用了机器描述[9]的方式来对目标机的资源进行描述, OPEN64 也是如此. 为了提供对 BWDSP10x 谓词的支持, 则需要添加 BWDSP10x 谓词指令的描述. BWDSP10x 谓词部分的机器描述包括地址谓词 U/V/W 描述和数据谓词 CPred 描述两部分, 它们主要由以下几部分构成:

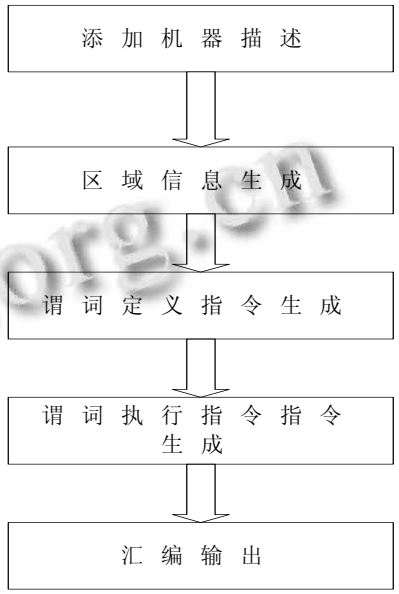

图 1 BWDSP10x 谓词执行优化框架

(1) 操作码类型

opcode += "<operate id>, <operate code name>, <function class in KAPI or fuDUMMY>,, <subset>, <opopndsgrp id>, <others>";

(2) 执行单元属性

unitprop[opname] := bitmask(eun_t);

(3) 寄存器类属性

regclassprop += "<item of REGCLASS>, <isa_mask>, <register number>, <bit_size>, <can_store>, <multiple_save>, <name>, <naming_rule>, <name of array holds special register name or null>";

(4) 寄存器子类

regsubclassprop += "<register subclass item>, <register class>,<member number list>, end(-1)";

(5) 寄存器 abi 和它的属性

TYPE ABIPROP = enum(entry_ptr, allocatable, callee, caller, stacked, func_arg, func_val, frame_ptr, stack_ptr, dedicated_int, dedicated_addr, true_predicate, ret_addr, prev_funcstate, loop_count, epilog_count, subprogram_ptr, branch_addr);

(6) 操作码属性

opprop :array[string] of bitmask(oppt);

(7) 操作数属性

opndtype [<operand types enum name>] := "<rclass>, <rsubclass>, <lclass>, <eclass>, <size>, <IS_signed>, <IS_fpu_int>, <IS_pcrel>";

opndsgrp += "<opndsgrp id>,<op specifier><operand types>/<uses name>";

3 区域信息生成

区域是一个流图中只有单个入口结点的部分, 用区域可以很方便地进行控制流优化. 区域有 5 种, 分别是: 根区域、多入口多出口区域、单入口多出口区域、不可规约区域、循环区域, 其中根区域是区域树最外层区域[10]. 一个自然循环就是一个区域, 但区域不一定包含一条回边. 为了构造区域, 必须找到自然循环, 然后分析它的区域结构. 任何两个自然循环要么不相交, 要么一个循环嵌套在另一个循环里. 没有循环的基本块本身就是一个区域. 一个程序的流图分为可规约流图和不可规约流图. 对于可规约流图, 把每个基本块当做一个区域, 把循环从内到外排序, 对每一个自然循环构造循环体区域和循环区域, 完成对循环的规约, 规约完成后所有的循环都被规约为单个结点. 规约过程: 首先处理自然循环, 直到流图中包含环没有回边, 剩下不可规约的部分. 不可规约的部分尽管可以用结点分割的技术成为可规约的, 但由于时间复杂度较高, 本文不对此进行处理, 但必须把用强连通分量算法[11]找出不可规约区域, 供谓词优化阶段优化使用. 对于重叠的循环, 考虑到时间复杂度, 不需要进一步优化. 最后, 对所有的区域进行分解提高目标机资源利用效率. 构建区域树算法伪代码, 如图 2 所示.

```
REGION_TREE(BB *first_bb) {
    Create_BB_Node_Map();
    REGION* _root = Add_Region();
    Build_Regional_Cfg(first_bb);
    _root.Process_Intervals();
    For each overlapped loop In REGION_TREE
        Do No_Further_Optimization();
    Decomposition();
}
```

图 2 区域树算法

算法输入为函数中的第一个基本块 first_bb, 输出为一个区域树, 树上每一个结点代表一个区域. 首先创建基本块结点映射, 把每个基本块映射到相对应的区域控制流结点上, 然后增加区域树的根结点, 根据全局数据流图构建局部区域控制流图, 在构建区域控制流图的同时计算相应的入口和出口信息, 最后处理循环和不可规约的部分以及重叠的非自然循环.

在构建区域树算法中处理循环和不可规约部分是在区域控制流图中进行的, 先找出环, 然后计算关键边和回边找到强连通分量, 构建循环区域和不可规约区域. 处理循环和不可规约部分算法伪代码如图 3 所示.

```
Process_Intervals() {
    Find_Cycles();
    Compute_Dominators();
    Collect_Backedges();
    Construct_Loops();
    Collect_Improper_Nodes();
}
```

图 3 处理循环和不可规约算法

尽管区域越大越好, 但如果区域超出了目标机的资源约束, 则会导致程序执行效率[6]降低. 因此, 区域树算法最后一步是按照目标机的体系结构对所有的区域进行分解, 把所有区域分解为较小的区域, 以最大利用目标机资源. 区域最终被分解为单入口多出口区域供后面谓词优化阶段使用. 区域分解算法伪代码如图 4 所示.

```
Decomposition() {
    If (region esources needed > target resources limited)
        Do Decompose_Region_To_SEME ();
}
```

图 4 区域分解算法

4 谓词优化算法

BWDSP10x 和 IA64 一样也提供了谓词指令, 但 BWDSP10x 和 IA64 有两点区别: 一、BWDSP10x 是指令和数据分开的哈佛体系结构, 而 IA64 是传统的冯•诺伊曼体系结构, 因此 BWDSP10x 支持 U/V/W 地址谓词和 CPred 数据谓词两种模式; 二、IA64 支持全谓词执行技术, 在指令执行前利用每条指令的限定谓词来决定此指令是否执行, 不需要单独的谓词执行指令,

而 BWDSP10x 只支持部分谓词, 需要单独的谓词执行指令.

BWDSP10x 的地址谓词寄存器有 16 个, 当做谓词寄存器使用时, 可使用其中的 8 位. 数据谓词寄存器即每一个执行宏上的一个独立的 8 位 CPred 寄存器. U/V/W 地址模式中把两个地址比较结果放在地址发生器 U/V/W 中的一个地址寄存器的一位中, 此时地址寄存器中存的是立即数. CPred 数据模式是把两个数据比较的结果放在一个 8 位谓词寄存器 CPred 中的一位中, 此时 CPred 中存的也是立即数.

BWDSP10x 两种谓词指令分别支持两种不同的谓词模式. 每一种指令都定义了谓词定义指令、谓词执行指令. 其中 U/V/W 模式还定义了谓词传输指令, 其原因是谓词定义指令的结果和源操作数都是地址, 而条件分支中的两个操作数之一可能是立即数或数据, 需要把其传输到地址寄存器中. CPred 模式中的谓词定义指令中的两个源操作数都是数据, 比较操作可以在通用寄存器中进行, 因此不需要谓词传输指令.

4.1 谓词定义指令生成

谓词定义指令生成是指把编译器中间代码生成的比较指令替换为谓词指令的过程.

BWDSP10x 的 U/V/W 地址谓词定义指令中的 U 地址指令形式如表 1 所示, V/W 地址形式类似, 这些指令均为单字指令. 其中, s、m、n 为地址谓词寄存器的编号. 指令含义为根据 Um 和 Un 的比较结果对 Us 的第 K 位置位. 若比较结果为真, Us 第[k]位置为 1, 否则置为 0, 0<=k<8. 本指令由 U 地址发生器中的“加法/移位”部件执行.

BWDSP10x 的 CPred 数据谓词定义指令中的指令形式如表 2 所示, 这些指令均为双字指令. 其中, x、y、z、t 为运算宏的编号, m, n 为通用寄存器编号. 指令含义为在四个运算宏之一内根据 Rm 和 Rn 的比较结果对 CPred 的第 k 位置位. 若比较结果为真, CPred 第 k 位置为 1, 否则置为 0, 0<=k<8. 第 k 个 ALU 控制 CPred 寄存器的第 k 位, 本指令由第 k 个 ALU 执行.

表 1 U/V/W 地址谓词定义指令形式

指令格式											汇编形式
31	30:27	26	25:23	22:18	17:16	15:12	11:8	7:4	3	2:0	Us[k]=Um>Un
行标志	0000	单/多	000	Op_code	模式	s 选择	m 选择	n 选择		K	

表 2 CPred 数据谓词定义指令形式

指令格式											汇编形式
31	30:27	26	25	24:21	20:18	17:15	14	13:12	11:6	5:0	{x,y,z,t}CPred[k]=Rm>Rn
行标志	运算宏	1	0	op_code	标志位 001	k		大小关系	Rm	Rn	
31	30:27	26	25	24:11			10	9:8	7:0		
行标志		1	1				符号	数据类型			

由上可知, 在生成谓词定义指令时需要对谓词寄存器每一位进行编号确定运算单元比较结果置位的位置. 两种谓词寄存器的谓词定义指令比较结果安排方式一样. 谓词寄存器的每一位表示一个比较条件, 8 位代表在一个函数中, 一个 if 语句中最多能有 8 个比较条件或者最多有 8 层的 if-else 嵌套, 在一个 if-else 中每个条件置位谓词寄存器的一位. 置位的顺序从第一个比较条件开始置位谓词寄存器最低位, 根据 BWDSP10x 指令集谓词执行指令, 当 k 为 0 时指令必须执行, 因此谓词寄存器最低位从 1 开始标号, 而不是 0. 谓词定义比较结果安排示意如表 3 所示.

表 3 谓词定义中比较结果安排

高位				低位			
8	7	6	5	4	3	2	1
P8	P7	P6	P5	P4	P3	P2	P1

谓词定义指令生成的算法流程和算法伪代码分别如图 5 和图 6 所示.

在生成谓词定义指令时, 在找到分支指令后, 判断分支指令里比较的两个操作数之一是否为地址来决定采用哪一种谓词模式. 如果两个操作数均为数据则生成 CPred 谓词定义指令, 若其中一个操作数为地址另一个操作数为数据, 则把为数据操作数传输到地址

寄存器中, 生成 U/V/W 谓词定义指令. 在生成 U/V/W 谓词定义指令和 CPred 谓词定义指令之前要分别初始化一个虚拟谓词寄存器 u0 和对应执行宏的 CPred, 并分别对两种模式条件分支进行计数, 计数变量为全局变量 ak、ck, 初值设为 0, ak、ck 同时指明了要置位的谓词寄存器位编号, 选择哪个执行宏上的 cpred 寄存器要根据操作数的执行宏来决定,当 ak、ck、以及要使用的谓词寄存器确定后就可以插入如上的谓词定义指令.

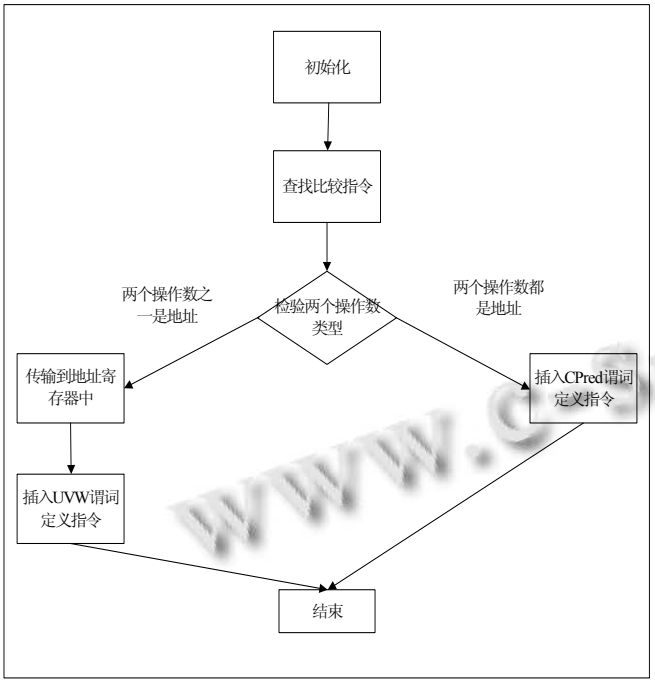

图 5　谓词定义指令生成算法流程图

```
ak=0;
ck=0;
u0=0;
cpred=0;
Pred_Def_Gen() {
    Find_Comp_Instr();
    if (Is_Address(first_opnd) || Is_Address(second_opnd)) {
        Transfer_To_Address(first_opnd);
        Transfer_To_Address(second_opnd);
        ak++;
        if (ak <= 8)
            Inster_ UVW_Def_Instr();
    }
    Else
    {
        ck++;
        if (ck <= 8)
            Inster_CPred_Def_Instr();
    }
}
```

图 6　谓词定义指令生成算法伪代码

4.2 谓词执行指令生成

在谓词定义指令生成后开始谓词执行指令的生成, 谓词执行指令的生成过程也是把跳转分支合并为单个基本块的过程. 在 IA64 中这是通过在跳转目标块前加上限定谓词实现的, 而在 BWDSP10x 采用的是生成谓词执行指令的方式实现.

BWDSP10x 的 U/V/W 地址谓词执行指令中的 U 地址指令形式如表 4 所示, V/W 地址格式类似, 这些指令均为双字指令. 指令判断 Um 对应 K1 为 1 位置的数据是否等于常数 C1 对应 K1 为 1 的位置的数据, 若判断结果为真, 则执行当前指令行中该指令后的前 n 条指令, 包括所有指令类型.

BWDSP10x 的 CPred 数据谓词执行指令中的指令形式如表 5 所示, 这些指令均为双字指令. 指令判断 CPred 对应 K1 为 1 位置的数据是否等于常数 C1 对应 K1 为 1 的位置的数据, 若判断结果为真, 则执行当前指令行中该指令后的前 n 条指令, 只控制这些指令中受 CPred 控制的指令. 本指令不能控制非运算指令、宏间传输指令、访存指令(无论读或写)等执行单元不在宏内的指令.

表 4　U/V/W 地址谓词执行指令形式

指令格式														汇编形式
31	30:27	26	25	24:21	20:18	17:16	15	14	13	12:9	8:5	4	3:0	
行标志	K2[7:4]	1	0	op_code	标志位 001	U/V/W 选择		判断模式 1	判断模式 2	m	i		n	If Um[K1]==C1 do n
31	30:27	26	25	24	23:16			15:8			7:0			
行标志	K2[3:0]	1	1		C2			K1			C1			

表 5 CPred 数据谓词执行指令格式

指令格式												汇编形式
31	30:27	26	25	24:21	20:18	17:15	14	13	12:8	7:4	3:0	
行标志	1111	1	0	op_code	标志位 010		判断模式 1	判断模式 2		K2[7:4]	n	If CPred[K1]==C1
31	30:27	26	25	24	23:16			15:8		7:0		do n
行标志	K2[3:0]	1	1		C2			K1		C1		

一个常见的带多个条件分支的区域通过谓词执行技术规约过程如图 7. 首先过程 1 中基本块 2, 3 合并到基本块 1 中, 7 合并到 6 中, 完成一次规约后删除条件分支, 转变为过程 2, 再进行规约删除条件分支最终转换为过程 3. 算法流程如图 8, 具体算法伪代码如图 9 所示.

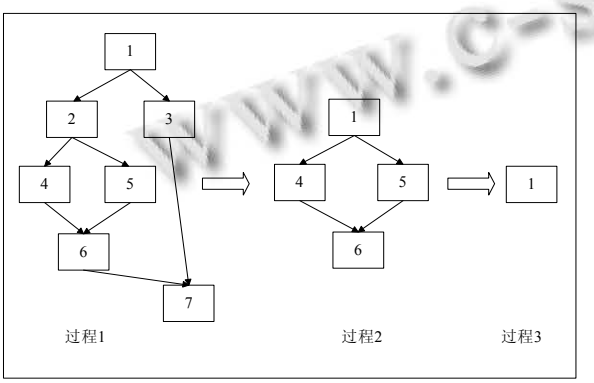

图 7 条件分支转换规约过程

```
Pred_Exec_Gen() {
    OP def = Find_Pred_Def_Instr();
    OP_code pred_op = Get_Op_Code(def);
    INT result_k = Get_OP_Result_k(def);
    if (Is_Pred_UVW(pred_op)) {
            Compute_Block_total_Ops();
            If(result_k == 2 ^ (result_k - 1))
              Inster_UVW_Exec_Instr();
    } else if (Is_Pred_CPred(pred_op)) {
            Compute_Block_total_Ops();
            Move_Not_Ctrl_Instr();
            If(result_k == 2 ^ (result_k - 1))
              Inster_CPred_Exec_Instr();
    }
    Remove_Branch_Instr();
    Merge_Branch_Target_Block();
    Merge_Branch_FallTrough_Block();
    Merge_Branch_Target_Succ_Block();
}
```

图 9 谓词执行指令生成算法伪代码

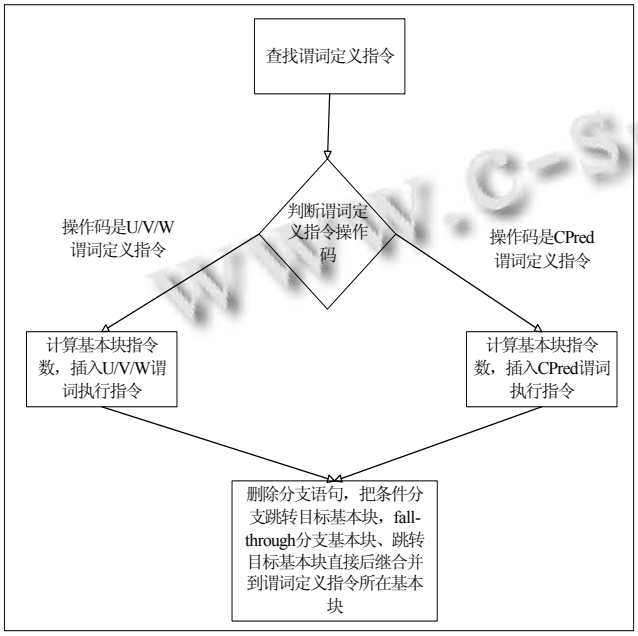

图 8 条件转换规约算法流程图

算法首先查找前面生成的谓词定义指令, 根据谓词定义指令的操作码决定是生成 U/V/W 地址谓词执行指令还是 CPred 数据谓词执行指令. 然后获得当前谓词定义指令中的谓词寄存器被置位的 k 和常量 2^(result_k-1)相比较得到条件比较结果对跳转目标或落入分支设置限定谓词, 即生成谓词执行指令. 在生成 CPred 数据谓词执行指令时, 需要把设置限定谓词的基本块中不受 CPred 控制的指令移动到谓词执行指令的前面; 而且, 由于在 CPred 数据谓词定义指令生成时已经确定了执行宏, 此时不再需要重新确定执行宏.

5 实例分析

常见的地址比较条件分支程序优化前和优化后对比如图 10 所示.

常见的数据比较条件分支程序优化前和优化后对比如图 11 所示.

```
Address a,b,c;                  U0=a;
if(a>=b)                        U1=b;
 c=a+50;                         ak++;
else                            U7[ak]=U0>=U1;
 c=b+70;                        If U7[ak]==2^(ak-1) do 1;
                                U6=U0+50;
                                If U7[ak]!= 2^(ak-1) do 1;
                                U6=U1+70;
```

优化前　　优化后

图 10　地址比较条件分支优化前后对比

```
Int a,b,c;                      xr39=a || xr38=b || xr36=50 || xr35=70;
if(a>=b)                        ck++;
      c=a+50;                   xCPred[ck]=xr39>=r38;
else                            if CPred[ck]==2^(ck-1) do 1;
      c=b+70;                   xr37 =r39+r36;
                                if CPread[ck]!=2^(ck-1) do 1;
                                xr37=r38+r35;
```

优化前　　优化后

图 11　数据比较条件分支优化前后对比

实验选取了 SPEC CPU 2006 测试项目中的五个程序来验证在打开谓词优化选项后的优化结果. 这五个程序分别是压缩程序 bzip2、组合优化 mcf、寻路算法 astar、有限元分析 dealII、影像光线追踪 povray, 这些程序在 DSP 领域应用很广泛.

这五种程序单独实施地址谓词优化结果如表 6 所示.

表 6　实施地址谓词优化前后对比

程序(N=1000)	优化前时钟数	优化后时钟数	性能提升(%)
压缩程序 bzip2	5038	4383	13.0
组合优化 mcf	4597	3838	16.5
寻路算法 astar	3692	3020	18.2
有限元分析 dealII	12536	10305	17.8
影像光线追踪 povray	13728	11037	19.6

这五种程序单独实施数据谓词优化结果如表 7 所示.

表 7　实施数据谓词优化前后对比

程序(N=1000)	优化前时钟数	优化后时钟数	性能提升(%)
压缩程序 bzip2	5038	4675	7.2
组合优化 mcf	4597	4202	8.6
寻路算法 astar	3692	3397	8.0
有限元分析 dealII	12536	10894	13.1
影像光线追踪 povray	13728	11902	13.3

这五种程序同时实施两种谓词优化结果如表 8 所示.

表 8　同时实施两种谓词优化前后对比

程序(N=1000)	优化前时钟数	优化后时钟数	性能提升
压缩程序 bzip2	5038	3980	21%
组合优化 mcf	4597	3388	26.3%
寻路算法 astar	3692	2658	28.0%
有限元分析 dealII	12536	8562	31.7%
影像光线追踪 povray	13728	8895	35.2%

从以上实验结果可以看出, 表 6 中单独实施地址谓词优化程序性能平均提升了 17%; 表 7 中单独实施数据谓词优化程序性能平均提升了 10%; 从表 6 和表 7 的结果可以看出, 由于增加了地址谓词的优化, 使对地址的比较不用传输到通用寄存器中再进行比较, 节省了时间和带宽, 比传统的单一谓词优化模式性能有很大提升; 从表 8 可以看出寻路算法和影像光线追踪优化效果要比压缩程序好, 说明谓词优化技术特别适用于地址比较条件分支和数据比较条件分支较多的程序, 表 8 中同时实施两种谓词优化程序性能平均提升了 28.4%, 说明实施谓词优化后消除了一部分分支指令减少了流水线停顿使指令流水效率大大提高了.

6　结语

本文针对 BWDSP10x 体系结构提供的谓词指令, 提出了一种条件分支转换的方法. 通过生成谓词定义指令和谓词生成指令, 使源程序的条件分支基本块可以合并为一个基本块, 充分提高了程序的并行性. 这种方法适合指令和数据分开的哈佛结构, 能对地址和数据的条件分支优化分开进行, 进一步提高了程序的效率. 本文只对两种谓词模式的生成做了研究, 接下来还应在生成的谓词指令的基础上进行各种数据流优化分析.

参考文献

1 冯玉谦.基于多簇 VLIW 软件流水及相关编译优化技术研究[硕士学位论文].合肥:中国科学技术大学,2013.

2 付和萍.基于超长指令字的全局无环指令调度和复数乘法优化设计[硕士学位论文].合肥:中国科学技术大学,2013.

3 黄伟,王玉艳,章建雄.嵌入式处理器动态分支预测机制研究与设计.计算机工程,2008,34(21):163–165.

4 Gary ST. The effects of predicated execution on branch

prediction. Proc. of the 27th Annual International Symposium on Microarchitecture. 1994. 196–206.

5 芦运照.谓词相关编译技术和深层代码优化[博士学位论文].北京:中国科学院研究生院计算技术研究所,2004.

6 刘旸.基于区域的编译技术和栈寄存器优化[博士学位论文].北京:中国科学院研究生院计算技术研究所,2003.

7 CETC. BWDSP100 软件用户手册.合肥:中国电子科技集团公司第三十八研究所,2013:1–2.

8 Aho AV, Lam MS, Sethi R, et al. Compilers: Principles, Techniques, and Tools. 2nd ed., Beijing: China Machine Press, 2009: 428–436.

9 杨萍,王生原.用于多目标编译系统构造的目标机体系结构描述.计算机科学,2005,32(9):239–242.

10 刘旸,张兆庆,乔如良.基于域的编译框架.计算机学报,2003,26(2):190.

11 Lengauer T, Tarjan R, Endre R. A fast algorithm for finding dominators in a flowgraph. ACM Trans. on Programming Languages and Systems, 1979, 1(1): 121–141.